COMP9517 Computer Vision
Assignment Report

Student: Rangbin Li (z5155841)

Task 1

For this task, the aim is to convert a color image to a grey-level image by doing some operations on each pixel. The tests images are *dog.jpg* and *light_train.jpg*.

Here are the details of the implementations. First, I read an input color image as A. Because it is BGR type so I use *cv2.cvtColor()* to convert it to RGB type. Then I implement a function called *rgb_to_gray(rgb)*, which can calculate the dot product of two arrays. The required formula is:

$$I(x,y) = 0.299 * r(x,y) + 0.587 * g(x,y) + 0.114 * b(x,y)$$

I use this function instead of a nest-loop because I thought it could save some processing time. Then I simply set the dtype of the image to uint8 and get the result picture named I by saving the images using *cv2.imwrite()* function. The images are showed below.

original

result

original

result

The two picture named Results are the gray scale images. Although it is hard to find out the exact difference between the original and the result pictures, many features have been lost since the color bands was reduced to one.

Task 2

For this task, the aim is to create a gray scale oil painting picture based on the result from Task 1. And we need to choose three different window size to generate the painting.

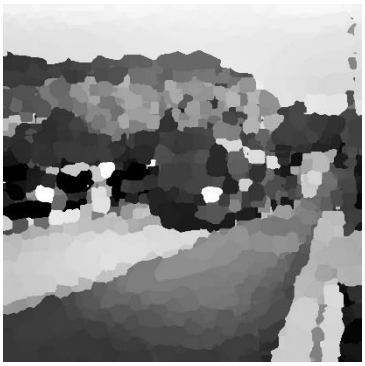
Here are the details of the implementations. First, I use *np.copy()* function to generate a copy of image I and named the new picture J. I cannot simply type *J = I* because it is something like a pointer in C language. When I operate J, I will be operate simultaneously. Then I chose three window size, 3, 20 and 50 respectively. The basic logic is to iterate over every index of pixel and find the most frequent pixel value and use this value as the pixel value of current location.

Here, I calculated the histogram in every window and pick the maximum value out and set it as the pixel value of current location. I use *cv2.calHist()* to calculate the histogram and *np.argwhere()* to find the index of the maximum histogram, which is also the most frequent pixel value of that specific area. The result pictures can be viewed below.



window size 3 *window size 20* *window size 50*





window size 3 *window size 20* *window size 50*

From the above pictures we can know that for the window size of 3, both two pictures look like the original ones. It is hard to notice big differences from old ones. With the window size of 20, the differences become more clearly. And with the window size of 50, we can observe a lot of pixels are just different from they are used to be. However, I will choose the window size of 20 as a real oil-painting because it looks better than the others. Other pictures are operated excessively.

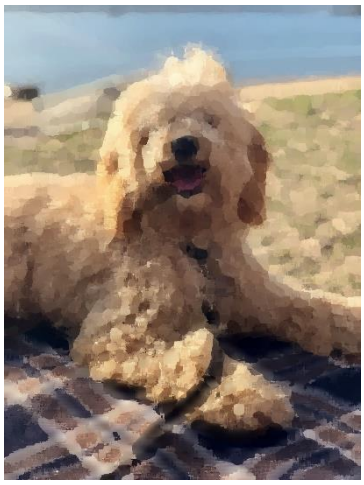
Task 3

For this task, the aim is to create an oil painting picture in colors based on the gray oil painting pictures generated on Task 2. We will use the same three window size as Task 2.

Here are the details of the implementations. First, I copied the image A to a new image B. Then I implement a function called *get_color_indensities* to calculate the average intensities of those pixels in each band. Find the current pixel value and create three empty lists to store pixel values. Then iterate the window and

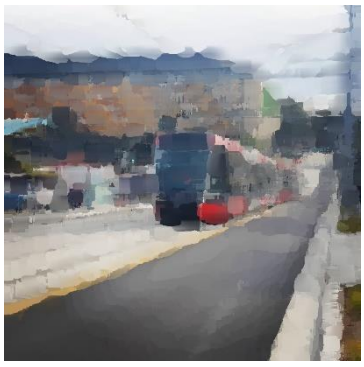
find out the same pixel values and stored in the list. Finally calculate the average. Finally, I used a nest-loop to set this pixel value to current location and generate the final oil painting. Below are the result images.





window size 3 *window size 20* *window size 50*





window size 3 *window size 20* *window size 50*

As for this part, the larger window size is, the slower program works. Just as Task 2, I think window size of 20 works the best.

Notes

To run the program, the only variable that need to be changed is the image name. I put this section to a separate part called *Variables That Need to be Changed* in the jupyter file. I also use formatted output name for different images and different window sizes, which should be easy to distinguish.